LISP, Lore and Logic: An Algebraic View of LISP Programming, Foundations, and Applications. W. Richard Stark. 278 pp. Springer-Verlag, 1990. $38.

Every science has its own vocabulary, but computer science is unusual in having whole languages of its own; indeed, every subculture of the computing world seems to have a private language. Those who specialize in numerical analysis tend to write in FORTRAN; the authors of operating systems favor C; other groups communicate among themselves in languages such as Pascal, Ada, FORTH, SNOBOL and APL. One of the oldest computing cultures—it goes back to before 1960—is the one that speaks LISP. This language has traditionally been associated with the pursuit of artificial intelligence, but it is a general-purpose programming notation, and it need not be confined to any one realm of discourse.

W. Richard Stark's book promises, among other things, the *lore* of LISP, and there is a lot of lore to be recorded. For example, the most conspicuous elements of any LISP utterance are the parentheses, as in this expression:

```
(car (cdr (cdr '(foo bar baz))))
```

It turns out that the heavily parenthesized notation was invented early in the development of the language as a kind of stopgap, to be used only until something better could be devised. In 30 years nothing better has come along. Those odd terms `car` and `cdr` also embody a good deal of history; they are verbal fossils, referring to hardware features of a vacuum-tube computer that was retired early in the 1960s. Their meaning is a little less abstruse than their etymology: `car` selects the first element in a list of items, and `cdr` selects all the elements except the first. These actions combine in such a way that the expression given above selects the third item in the list `(foo bar baz)`, namely the symbol `baz`. (Exploring the origin of the nonsense words `foo`, `bar` and `baz` would take us even deeper into LISP lore.)

Are these historical arcana worth knowing? It is certainly possible to write LISP programs without knowing the origin of `car` and `cdr`, just as one can write English prose without knowing about the Indo-European roots of the words. But the historical background of LISP, the circumstances of its invention, and its conceptual foundations are of more than passing interest. LISP is not merely a programmer's tool or a medium of communication but also an object of study in its own right.

At the heart of LISP is a model of computation closely related to the mathematical principles of recursive function theory. A subset of LISP called pure LISP is a strictly functional language: expressions in pure LISP have the simple semantics of mathematical functions, and so it is easy to reason rigorously (and even automatically) about programs written in this notation. A dialect of LISP called Scheme can be regarded as an implementation of the lambda calculus, the computational system invented in the 1930s by Alonzo Church. The rich body of mathematical results obtained in the lambda calculus can therefore be applied directly to Scheme; what is more, Scheme has become an excellent vehicle for teaching the ideas of the lambda calculus, since all the mathematical expressions can be evaluated by machine.

Most of the older texts on LISP pass quickly over the history and the theoretical foundations of the language. Stark gives ample attention to these matters, which is a welcome change. Bits of history and lore are scattered throughout the book, along with photographs of some of the leading figures and excerpts from important early works. There are discussions of the lambda calculus and other mathematical ideas, such as the theory of fixed points of functions.

Unfortunately, this material is not made accessible to those readers who might find it most informative. Somewhere within the book is most of what a beginning student needs to know, but there is no predicting just where in the book it will be found; in particular, there is no reason to expect that what you need to know first will be presented first. For example, the term "s-expression" is first used on page 31 but is not defined until 40 pages later. What is most disappointing is that the attractive mathematical properties of pure LISP are ignored in the introductory chapters; instead the student is introduced to methods of imperative programming (based on assignment statements) that would be more appropriate in a language such as Pascal or FORTRAN.

Apart from an abundance of parentheses, the most famous characteristic of LISP is probably its emphasis on recursive and self-referential structures. It may be apt, then, that Stark's book is most useful for learning exactly those things that you already know.—*Brian Hayes*